WANT MAX? ASK HEAP

Problem Solving with Computers-II

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```

Make a copy of the handout for today's lecture

https://bit.ly/cs24-lect14-handout

What is **mystery** doing ? (2 min)

```
void mystery(vector<int>& v){
    int n = v.size();
    for (int i = 0; i < n; i++){
        int index = i;
        for (int j = i + 1; j < n; j++){
            if(v[j] > v[index]){
                index = j;
            }
        }
        if(index != i){
            int temp = v[index];
            v[index] =  v[i];
            v[i] = temp;
        }
    }
}
```

finding the max

swap

Example input:

20	5	7	1	3	2

0

20 5 7 1 3 2

20 7 5 1 3 2

20 7 5 3 1 2

20 7 5 3 2 1

What is the time and space complexity of mystery? (2 min)

```
void mystery(vector<int>& v){
    int n = v.size();
    for (int i = 0; i < n; i++){
      find max of vector: v[i:n]

      swap v[i] with max element

    }
}
```

$O(n)$
$+$
$O(1)$

Running Time = $n \cdot (O(n) + O(1))$

$= O(n^2)$

Space Complexity:

$O(1)$

Brainstorm ideas to improve the running time. (3 min)

```
void mystery(vector<int>& v){
    int n = v.size();
    for (int i = 0; i < n; i++){
        int index = i;
        for (int j = i + 1; j < n; j++){
            if(v[j] > v[index]){
                index = j;
            }
        }
        if(index != i){
            int temp = v[index];
            v[index] =  v[i];
            v[i] = temp;
        }
    }
}
```

Notice that we are repeatedly finding the max!

```
void mystery(vector<int>& v){
    int n = v.size();
    for (int i = 0; i < n; i++){
        find max of vector: v[i:n]

        swap v[i] with max element

    }
}
```

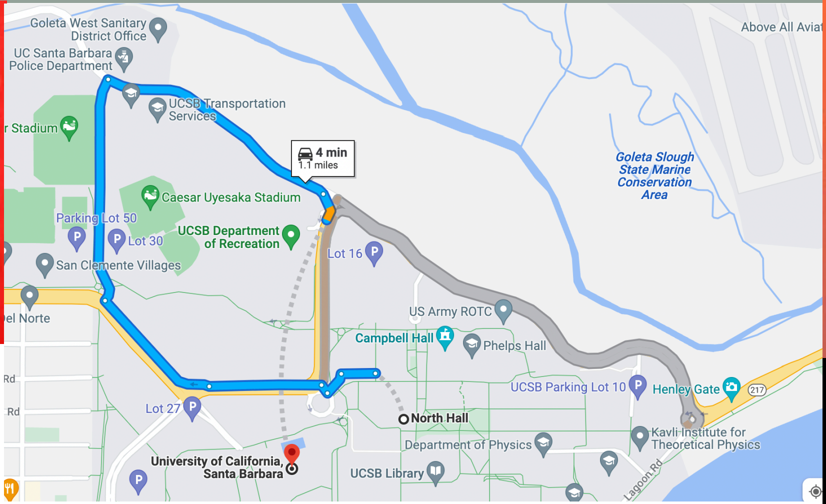
O(n)

O(1)

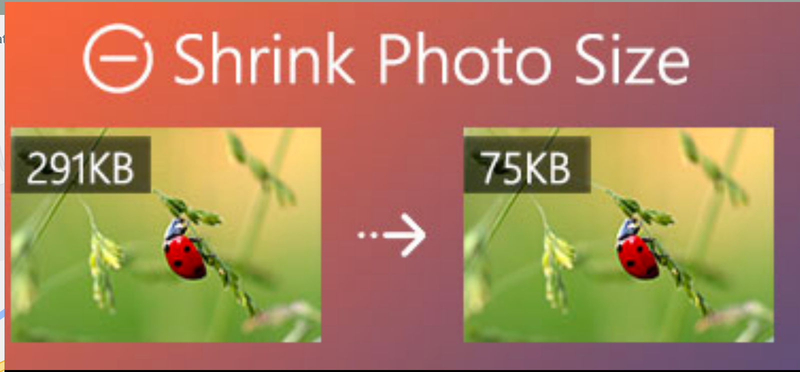
fast in a heap O(1)

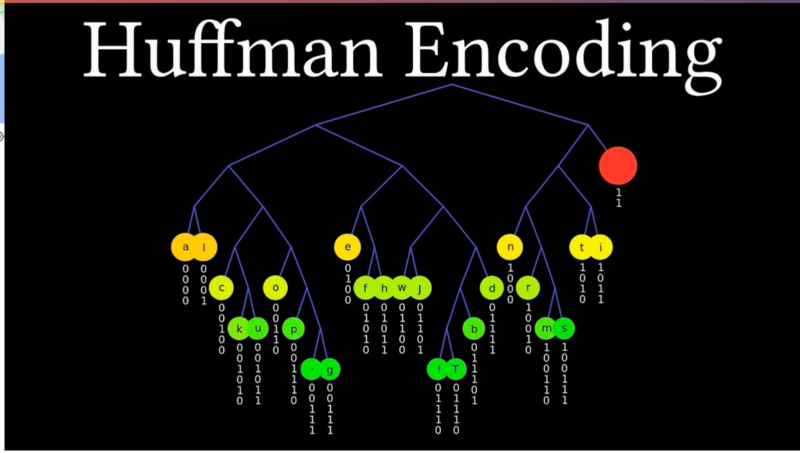
Insert + deletions O(log n)

Sorting


Goleta West Sanitary District Office
UC Santa Barbara Police Department
UCSB Transportation Services
4 min
1.1 miles
Stadium
Caesar Uyesaka Stadium
Parking Lot 50
Lot 30
UCSB Department of Recreation
Lot 16
San Clemente Villages
el Norte
US Army ROTC
Campbell Hall
Phelps Hall
Goleta Slough State Marine Conservation Area
Above All Aviat
UCSB Parking Lot 10
Henley Gate
217
Lot 27
North Hall
Department of Physics
Kavli Institute for Theoretical Physics
University of California, Santa Barbara
UCSB Library
Lagoon Rd
Rd
Rd

Shortest Path





Data Compression

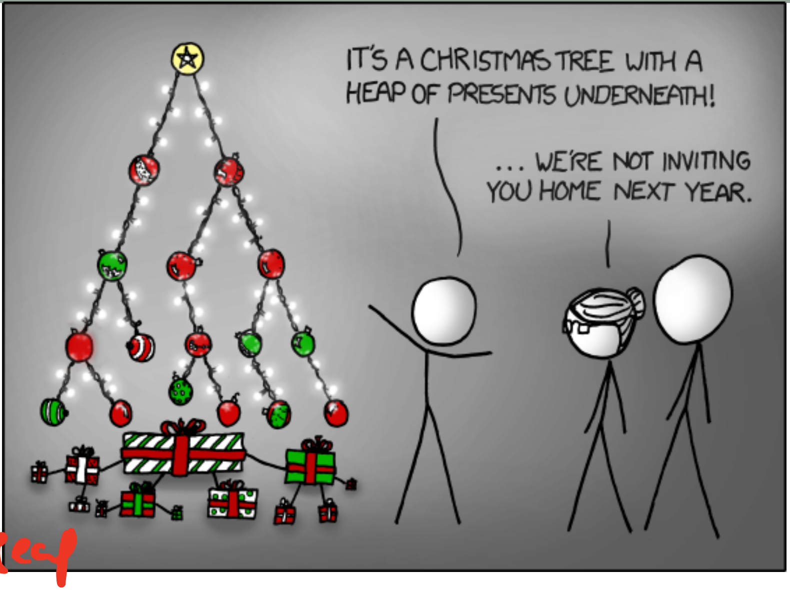
Want min or max? Ask Heap!

Many algorithms need to compute the min OR max repeatedly

Heap is used speed up the running time of such algorithms!

New data structure: Heap

- Clarification
  - *heap*, the data structure is not related to *heap*, the region of memory
- What are the operations supported?
- What are the running times?


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.

max-heap

min-heap

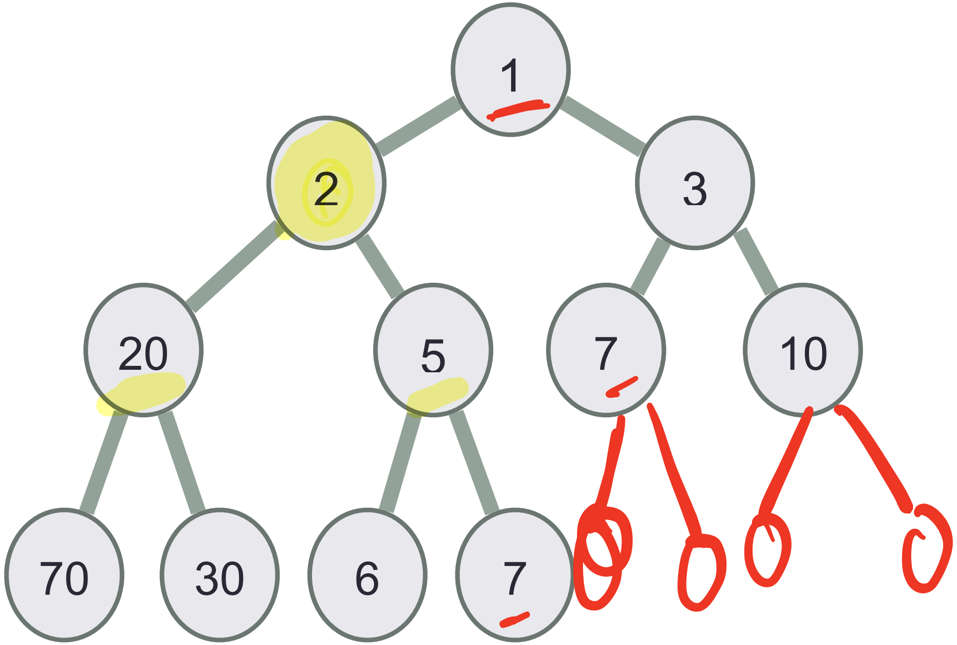
top() // min OR max O(1)

push(x) // insert O(log n)

pop() // deletes on the top O(log n)

Two important properties of a heap

Height = $O(\log n)$



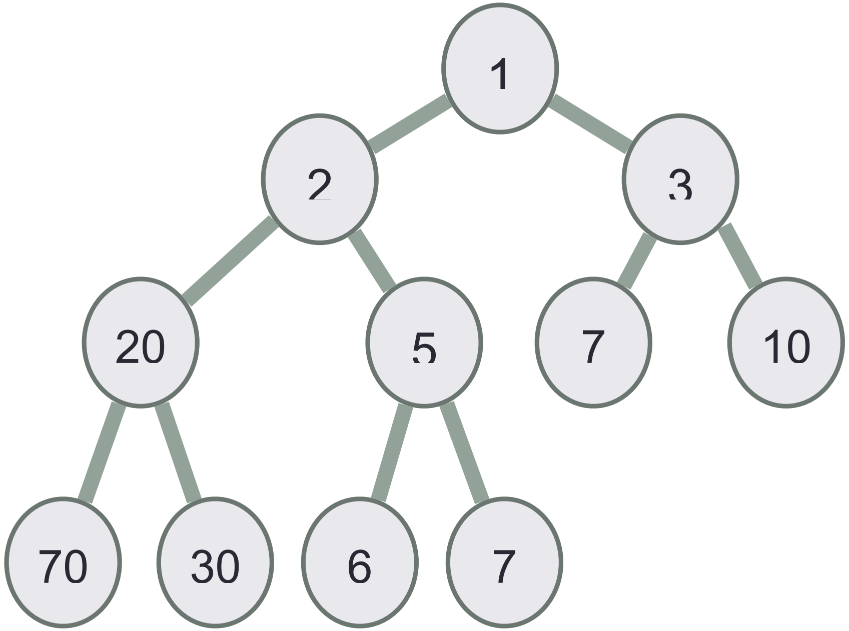
Shape property:

Complete binary tree

Heap property :

min Heap : for every node x $key(x) \le children(x)$

max Heap : $key(x) \ge children(x)$



Shape property:

Internally, a heap is a **complete binary tree,** where each node satisfies the heap property

Heap property :

In a min-heap, for each node (x):

key(x) <= key(children of x)

In a max-heap, for each node (x):

key(x) >= key(children of x)

Identifying heaps

Starting with the following min-Heap which of the following operations will result in something that is NOT a min Heap

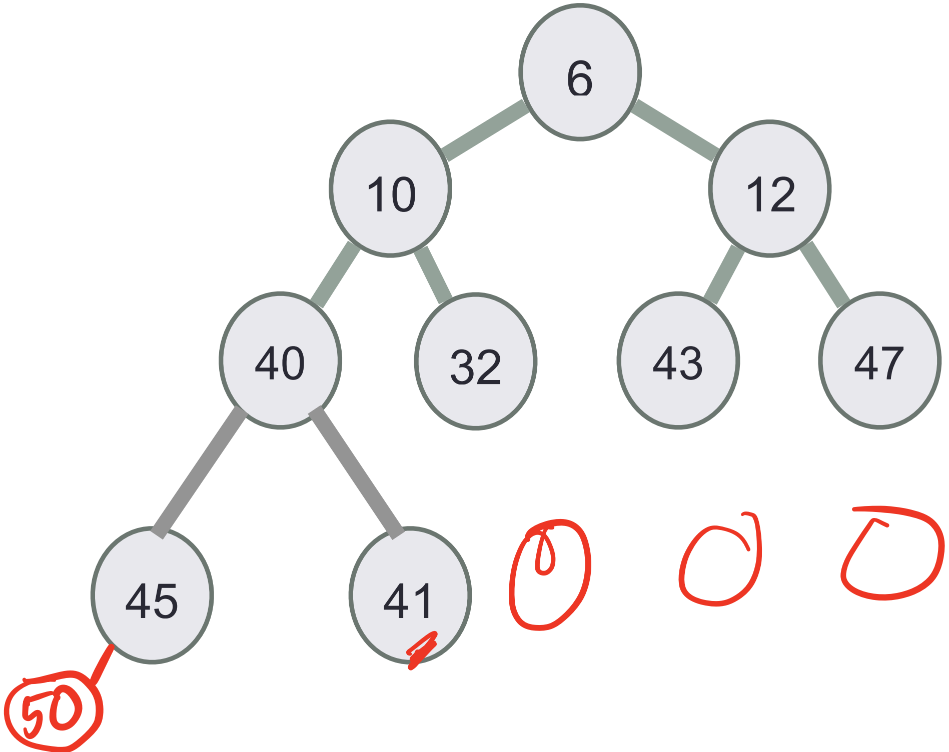
✓ A. Swap the keys 40 and 32

✓ B. Swap the keys 32 and 43

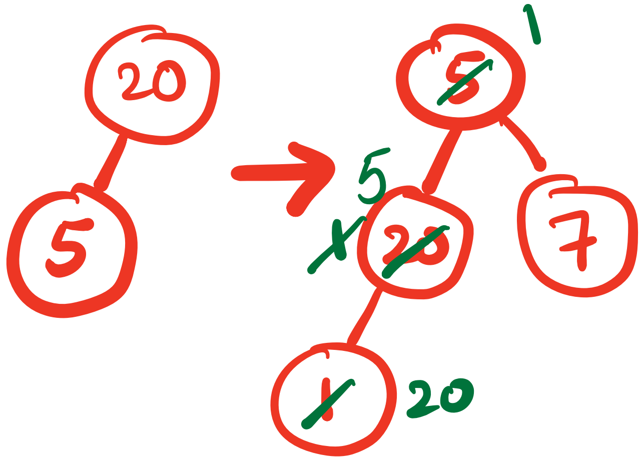
✗ C. Swap the keys 43 and 40

✗ D. Insert 50 as the left child of 45

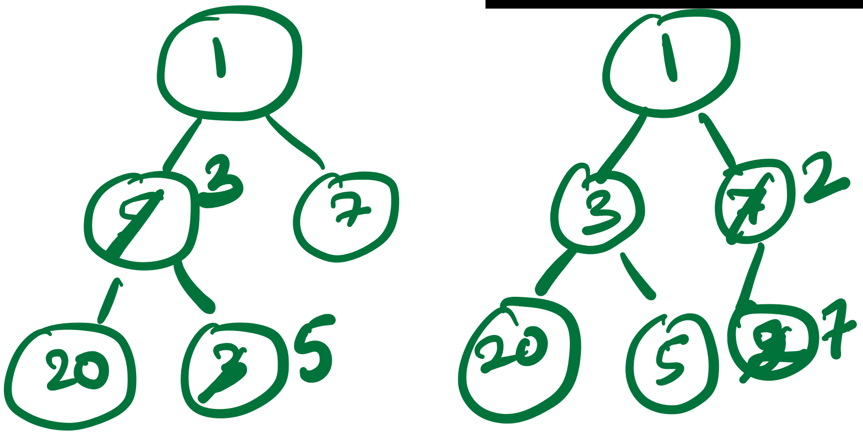
E. C&D



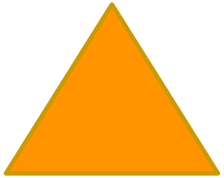
20, 5, 7, 1, 3, 2
push(x)



push(20)
push(5)
push(1)

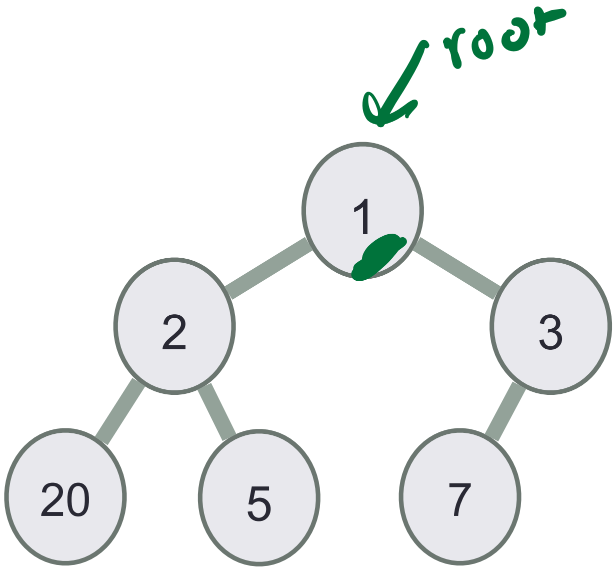


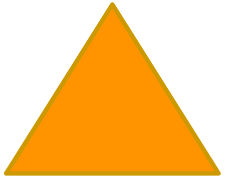
Overall O(log n)



min-Heap

```
procedure push(x: key value)
    insert x in the first open spot in the tree        O(1)    // Preserve the shape property
    while(x has a parent && x < parent(x)):            // Bubble up.
        swap(x, parent(x))
    return {x was inserted into a min-heap}
```

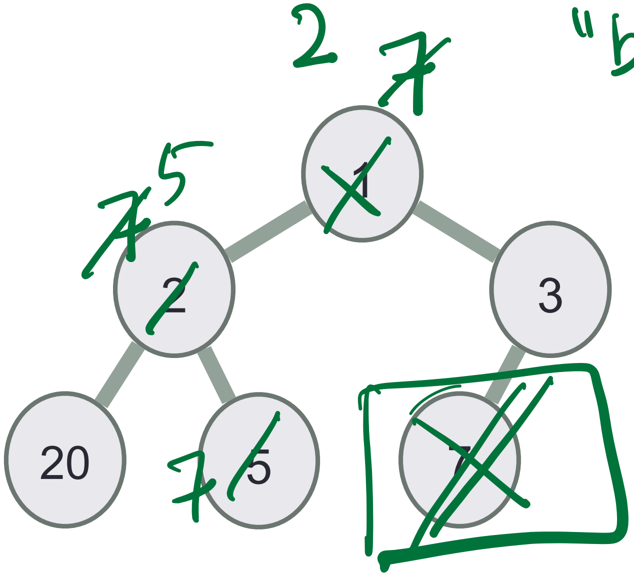




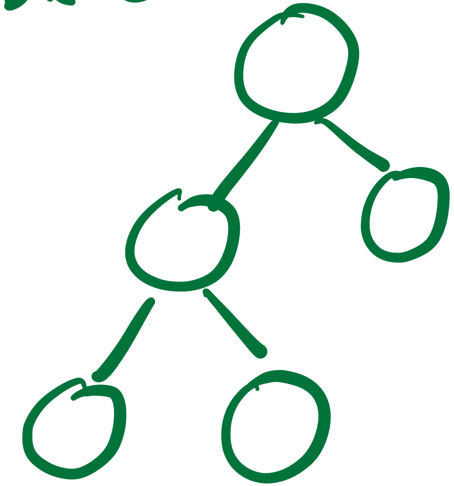
min–Heap

O(1)

```
procedure top()
  return key of root node {top element is returned}
```



"bubble down"



20, 5, 7, 1, 3, 2

pop()

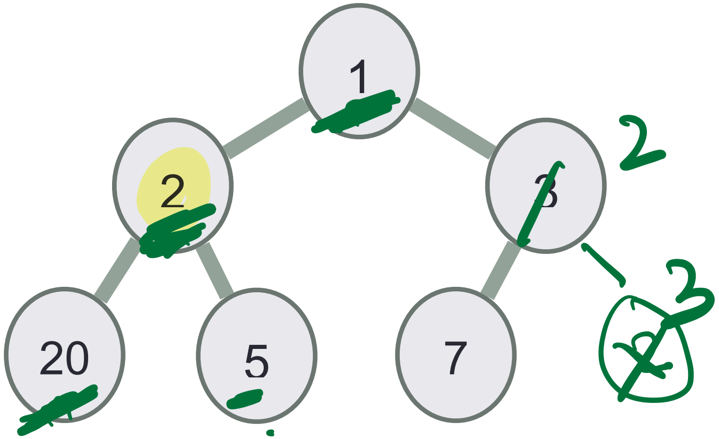
delete key at the top

```
procedure pop()


  return {key on top of the heap is deleted}
```

Internally the “heap binary tree” is really just a vector!

Breadth First Traversal of the tree



Internal representation V

i 2 3

	1	2	~~3~~	20	5	7	~~3~~	
Index of key	0	1	2	3	4	5	6	
Index of parent	-1	0	0	1	1	2		
Index of left child	1	3	5	-	-	-		
Index of right child	2	4	-	-	-	-		

key at index i

index of the parent $\frac{(i-1)}{2}$

index of the left child → $2i+1$

right child → $2i+2$

Work to complete the table on page 5 on your handout

Repeat the exercise on page 4 of your handout to insert the values 20, 5, 7, 1, 3, 2 into an initially empty min-heap. But instead of drawing the results as a tree, draw the resulting vector

```
procedure push(x: key value)
  insert x in the first open spot in the tree
  while(x has a parent && parent(x) > x):
    swap(x, parent(x))
  return
```

Next lecture

STL implementation of heap : priority_queue

Configuring priority_queue in different ways